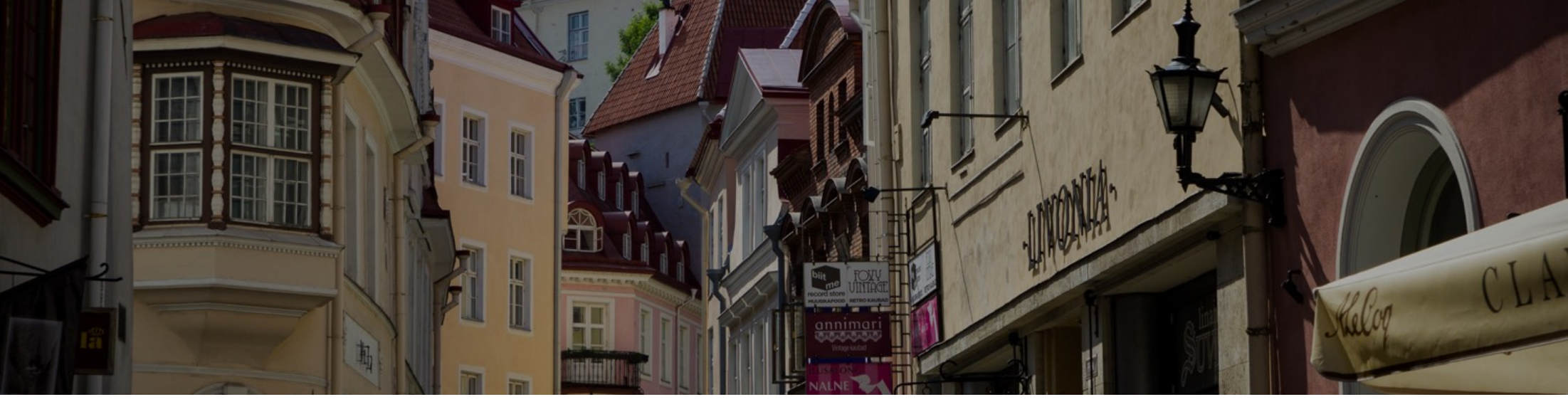
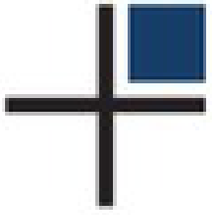


Advanced SQL with PostgreSQL

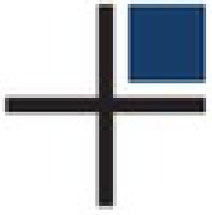
2ndQuadrant®
PostgreSQL

Gianni Ciolli

PGConf.EU 2016
Tallinn, 1-4 November

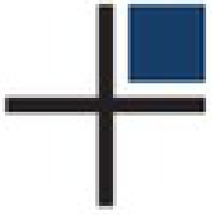


Outline

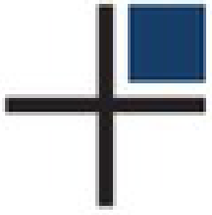


Topics

- SQL Language
- Advanced Constructs
  - INSERT . . . ON CONFLICT
  - Grouping Set
  - Transaction Isolation
  - Further examples

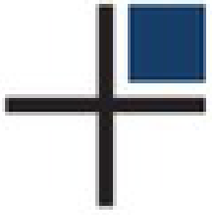


Outline



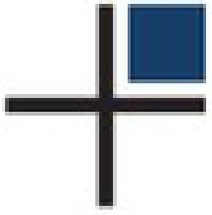
INSERT . . . ON CONFLICT DO . . .

- Implements "UPSERT"
  - "Either UPdate or inSERT"
  - Faster
  - Concurrent
  - Twice better!
- Two cases
  - . . . DO NOTHING
  - . . . DO UPDATE . . .



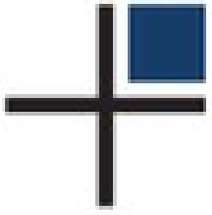
Example

```
INSERT INTO mytable(id,amount)
  VALUES (1, 100)

  ON CONFLICT(id) DO

  UPDATE SET amount =
    mytable.amount + excluded.amount;
```



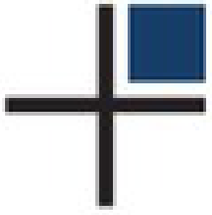
Grouping Set

- Summaries
- Plural!
  - Several summaries in a single query
- Similar to GROUP BY



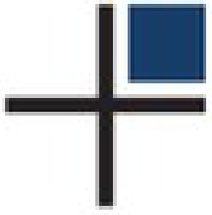
Example

```
SELECT brand
, customer
, sum(amount) AS amount
FROM orders
GROUP BY
GROUPING SETS ((brand), (customer));
```

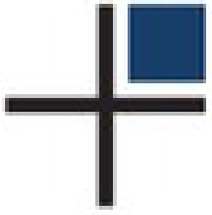


Example

brand	customer	amount
A		181468
B		208075
C		115369
	x	137483
	y	194331
	z	173098

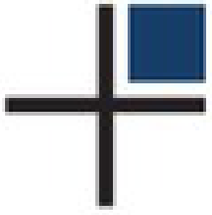


Outline



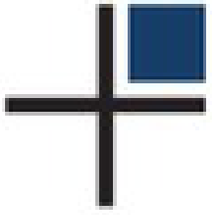
Serializable Transaction Isolation

- Advanced technique
- What is better:
  - Locking?
  - Serializable Transaction Isolation?
  - Other (specify)



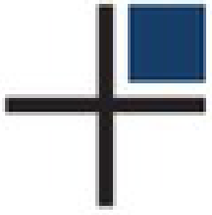
Lock

- Pessimistic by nature:
  - **lock** a resource so it **cannot be used** when inappropriate
- Simple to use
- Hard to get it wrong
  - (relatively hard)



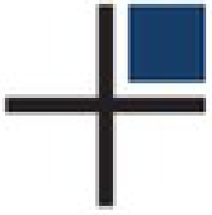
Lock

- Can unnecessarily lock **too much**
  - Example: (road) **semaphore**
- Performance loss
  - Not visible
  - Hard to detect / measure
- Row level locks are more selective
  - However they cause writes



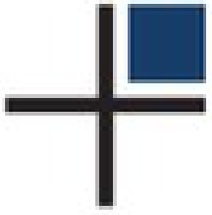
Serializable Transaction Isolation

- More **optimistic** than locking
- Idea: detect conflicts **afterwards**
- Hard to understand, but easy to use



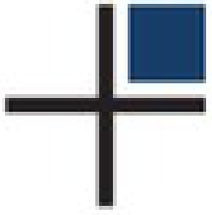
Serializable Transaction Isolation

- **Safe** anyway:
  - Mistakes result in applicative errors
    - Very visible
    - Very measurable
  - The application must be **adapted**
- No **false positives**
  - (To be honest: "not too many")
  - Better performance



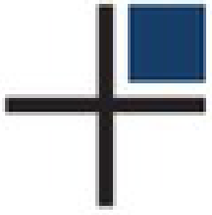
Abstract Example

```
BEGIN TRANSACTION
  ISOLATION LEVEL SERIALIZABLE;

  -- (query 1)

  ...

  -- (query N)

COMMIT;
```



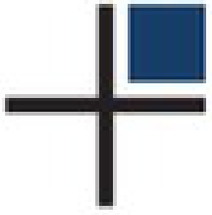
Technical Details

- Uses **Predicate Locking** to detect conflicts
- Predicate locks do not **block**
  - No deadlocks!
- They are lightweight
- When too many, escalated to keep them fast
  - Row Lock → Page Lock → Relation Lock
  - That’s the "not too many" from before



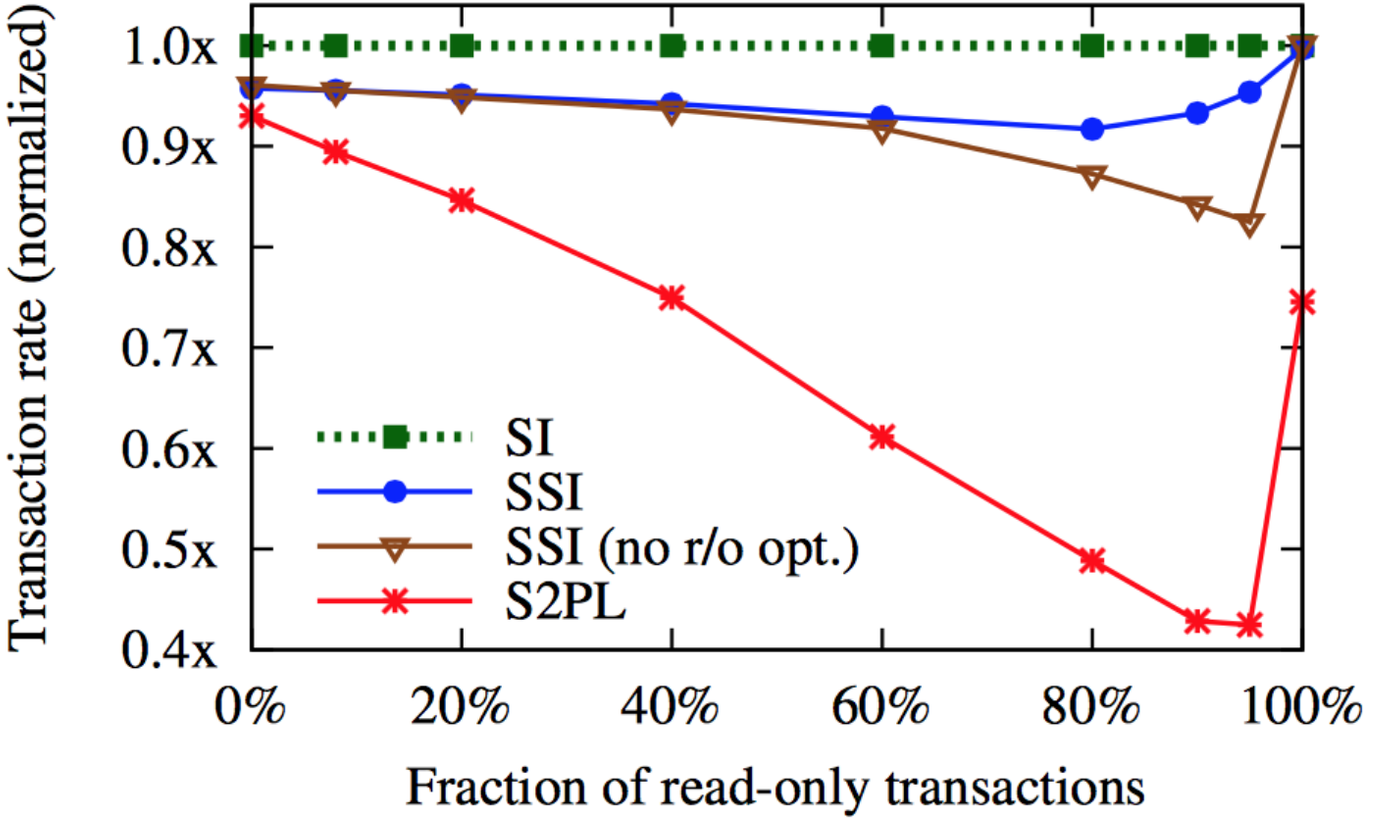
Predicate Locking Example

- Serializable transaction made by several queries
  $T' = Q_1, Q_2, \ldots$
- Another transaction which writes between Q_1 and Q_2
  $T'' = W_1$
- Q_1 takes predicate locks
- W_1 hits them
- T_1 is rolled back
- No hit, no rollback!

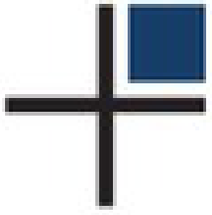


Predicate Locking Performance

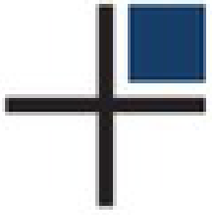
- From *Serial Snapshot Isolation in PostgreSQL*, D.Ports & K.Grittner, VLDB 2012



(a) in-memory configuration (25 warehouses)

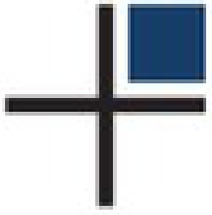


Outline



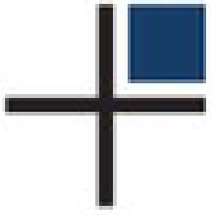
Further Examples

- Parallel Queries (since 9.6)
  - Parallel Aggregates
    - User defined, too
    - "map / reduce" *elephant style*
- Window Functions
  - Moving Aggregates
  - Combine different rows
  - Example (2011): differential equations
    - Science *elephant style*

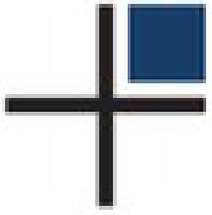


Further Examples

- Recursive Queries
  - Graph navigation
  - Graph database *elephant style*
- Move data atomically
- Etc.

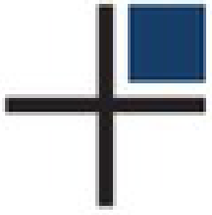


Outline



And Now. . .

Questions?



And Then...

Thank You!

gianni@2ndquadrant.com
@GianniCiolli

Feedback:

http://2016.pgconf.eu/f

Licence